CS109A Programming Project #1: Happy Medians

Due Friday, February 17, 1995

As you progress in computer science, you will see many facets of the field. This programming project gives you a tiny sample of three of these: a design side where, as in other engineering disciplines, you figure out how to build something; an experimental side where, as in other sciences, you observe and experiment with a system to understand its properties; and a theoretical side where you develop and use mathematical tools to clarify, justify, and inspire ideas from the other aspects of computer science.

An additional goal of this project is to give you experience working in teams. Although the project probably isn't big enough for teamwork to save a lot effort, the experience of working with and learning from your classmates is valuable. **You are permitted to organize into teams of up to three students each.**

This project is about computing medians. All the background material you should need follows. Details of what you are required to do are in the section, "Your Assignment."

Background

What's a Median?

Statisticians commonly use (at least) three different ways to summarize the "typical" or "middle" value of a set of numbers: the *mean*, the *median*, and the *mode*. The mean, commonly called the average, is what most of you are probably most familiar with. To compute the mean, just add up all the numbers and divide by how many there are. The median is the halfway number of a set — half the numbers are larger than the median, and half the numbers are smaller (precise definition below). The median is frequently a better measure of the typical or middle value of a set of numbers, especially for data in the social sciences or where there are a few outlandish errors in the data set. The mode is whatever number is most common in the set.

For example, suppose that California Weenie University graduated nine computer science majors last June. Four are currently unemployed, three are making \$20,000 working for Huge Computer Corp., one is making \$40,000 selling real estate, and one has a lucrative \$1,000,000 NFL contract. The mean annual salary for these people is

$$(4 \cdot 0 + 3 \cdot 20000 + 1 \cdot 40000 + 1 \cdot 1000000)/9$$

or about \$122,000. The median salary is \$20,000. And the modal salary is \$0.

Let's look more closely at medians, since that's the subject of this assignment. First, notice that we have to be careful about duplicate values. In the example above, for instance, since there are three $20,000$s, we don't actually have half the list larger and half the list smaller than the median we chose. The definitions given below handle duplicate values correctly. Also, note that if a list has even length, it's not immediately obvious what the median should be. For example, is the median of $[1, 2, 3, 4]$ the 2 or the 3? For this project, we will accept either answer. For an odd length list, the median is always unique. Here are the two precise definitions of the median of a list that we will use for this project:

Definition 1 *A number x is a median of a list L if and only if no more than half of the elements of L are strictly less than x and no more than half the elements of L are strictly greater than x.*

Definition 2 *A median of a list L is the ith smallest element of L, where $i = (length(L) + 1)$ div 2. If L has even length, then the $(i+1)$th smallest element of L is also a median.*

Algorithms for Finding the Median

The above definitions naturally suggest several algorithms for computing the median. Definition 1 naturally suggests:

Algorithm 1 *Given input list L, take each element of the list and compare it to every other element in the list, counting how many are strictly bigger and how many are strictly smaller than the given element. If both counts are no greater than half the length of the list, then the given element is a median.*

This algorithm will take $O(n^2)$ time in the worst case. Definition 2 suggests a faster approach: sort the list (time $O(n \log n)$), and then pick out the ith element of the sorted list (time $O(n)$). This approach would take $O(n \log n)$ time. But, there's an even faster method based on Definition 2 and related to QuickSort:

Algorithm 2 *Given input list L, compute $i = (length(L) + 1)$ div 2. Now, we recursively pick out the ith element of L as follows:*

1. *Partition L into two non-empty sub-lists — call them SMALL and BIG — such that every element in SMALL is less than or equal to the smallest element in BIG.*
2. *If $i \leq length(SMALL)$ then recursively select the ith element from SMALL.*
3. *Otherwise, $i > length(SMALL)$, so recursively select the $(i - length(SMALL))$th element from BIG.*

It turns out that if we can do the partition step in $O(n)$ time (which we can), and if the partition is into two nearly equal parts (which usually happens), then this algorithm will run in $O(n)$ time.

Your Assignment

OK, enough blabbing. What are you actually supposed to do? Your assignment is to write ML programs to compute the median of a list of integers using Algorithms 1 and 2. Next, benchmark your ML programs to determine the actual running time. Finally, analyze your ML programs and show that the big-oh time bounds we give you are true. Here are the details:

The Programming Portion

You are to write two ML functions that compute the median of an int list. The first function **must** be called `slow_median`, **must** take a single int list as its input parameter, **must** return a median of the list as the single int return value, and **must** be based on Algorithm 1. It should run in time $O(n^2)$. The second function **must** be called `fast_median`, take a single int list as its input parameter, return a median of the list as the single int return value, and be based on Algorithm 2. It should run in time $O(n)$. You should have both of these functions along with their assorted helper and sub-functions in a single input file. For example, when you're done, you should be able to type the following to SML:

```
- use "mymedian.sml";
[opening mymedian.sml]
   ...
val slow_median = fn : int list -> int
   ...
val fast_median = fn : int list -> int
   ...
val it = () : unit
- slow_median[3,1,4,1,6];
val it = 3 : int
- fast_median[0,0,0,0,20000,20000,20000,40000,1000000];
val it = 20000 : int
-
```

I strongly discourage you from trying to use the ML while loop or other unusual ML features that we haven't covered in class. You may use the built-in ML function `length` that returns the length of a list. For both `slow_median` and `fast_median`, I encourage you to build your solution out of a collection of small functions. For example, the actual function `slow_median` may just be a wrapper that takes the input in the specified form, calls a recursive function that does the actual work using whatever input and return values are more convenient for the recursion, and returns the result in the specified form.

For `fast_median`, the hard part will be partitioning the list in linear time. In particular, you must make sure that neither sub-list is empty, or else the program might not terminate. One way to ensure this is to use a three-way partition function that breaks the list into a lower list, a split element, and an upper list such that everything in the lower list is less than or equal to the split element, and everything in the upper list is greater than or equal to the split element. Then, the partition function can be just a wrapper that calls the three-way split function, and then adds the split element to whichever list is empty.

To help you test your programs, we've supplied a set of functions you may find helpful. This file is named "helpful.sml" in the class directory. In particular, this file defines a function `generator` that generates a random int list of the specified length, a function `printlist` that prints out an int list, and a function `grind` that you may find helpful for benchmarking. For example:

```
- use "helpful.sml";
```

```
[opening helpful.sml]
val random = fn : real
   ...
val it = () : unit
- generator 5;
val it = [~655330,~138132,~305639,~162594,128846] : int list
- printlist [1,2,3];
[
1,
2,
3
]
val it = () : unit
-
```

You are free to develop your program in any environment, but the finished program **must** run under SML/NJ on the Sweet Hall machines. Your program need not be long. My solution is under 100 lines, including blank lines and comments.

The Experimental Portion

After you've tested your median-finding function to convince yourselves they work (including on odd- and even-length lists, with duplicates, etc.), it's time to see how fast they run. For this portion, you should measure the running time of `slow_median` and `fast_median` on random lists of length 100, 200, 300, 400, ..., 1000. To smooth out irregularities in the running time, you should call each function at least 25 times on random lists of each size. Fortunately, we've made this easy for you by providing the `grind` function in the file "helpful.sml". For example, typing `grind(25,200,fast_median);` will call the function `fast_median` on 25 different random lists, each 200 elements long. You may thus time the call to grind, divide the resulting time by 25 and have the average (mean) running time for that input size.

By choosing the number of calls to make (25 should be fine, but you may need more) for each measurement, you should be able to measure the total time with a stopwatch. An alternative is the Unix time command. If you type `time sml < inputfile`, the machine will measure the time required to execute the command. Type `man time` at the Unix prompt for more information. Also, you are free to run your experiments under any environment, but again, the finished program must run under SML/NJ on the Sweet Hall machines.

When you are done, plot your results. Does your `slow_median` show quadratic running time? Does your `fast_median` show linear running time? If not, you should probably go back to the programming portion and fix your code.

You should realize that SML/NJ has a compiling phase, during which your program is processed but not run. This time adds a constant to the running time of your program that is not really due to the execution of your code. You can measure the compiler time by running your program on an empty list, and subtract it if you wish.

The Analysis Portion

Now that your code is behaving properly, it's time to see if big-oh analysis backs up your observations. For this portion, you should show that your `slow_median` runs in time $O(n^2)$ and that your `fast_median` runs in time $O(n)$.

For `slow_median`, your analysis should be for the worst-case. For `fast_median`, you can assume in your analysis that the partition step always splits the list perfectly into two equal-sized halves. In the worst case, that won't happen, but on average, the split is good enough to match the $O(n)$ bound you derive here. For both analyses, you may assume the built-in ML `length` function takes $O(i)$ time, where i is the length of the list you are calling it on.

What to Turn in and How

As mentioned earlier, your finished program must be able to run under SML/NJ on the Sweet Hall machines. You **must** turn in an on-line copy of your finished code as follows:

1. Choose one of your team member's leland account name to be the filename for your finished solution.

2. Log onto one of the Sweet Hall machines and copy your finished program using the name you chose in the preceding step into /usr/class/cs109a/dropoff.

For example, suppose My Hoang, Alan Hu, and Jeff Ullman are teammates. Suppose we've named the file that contains our median-finding functions "mymedian.sml". To turn in the on-line copy of our program, we can pick one of our leland account names (like "ajh" for Alan) and type:

```
cp mymedian.sml /usr/class/cs109a/dropoff/ajh
```

After you've done this, you will not be able to modify, read, or delete the file in the dropoff directory. You will, however, be able to use `ls` to see that your file got copied successfully. In addition, you need to turn in:

1. The name you used for your on-line file in the dropoff directory.

2. The program listing for your median-finding functions and all helper functions you wrote. Don't include a listing of "helpful.sml".

3. Your graph(s) of the running time of your functions.

4. Your analysis that shows that your `slow_median` function is $O(n^2)$ and that your `fast_median` function is $O(n)$.

This write-up should have all team members's names on it.